REMARKS

This response to the Office Action mailed on March 23, 2004 cancels no claims, adds no claims, and amends claims 1, 12, 16, 21, 25, and 29. As a result, claims 1-31 remain pending in the Application.

Drawing Objection

The drawings were objected to for not showing reference numeral "700" described in the Specification. A full set of formal drawings is submitted herewith, including Replacement Sheets 11 and 12 containing Figs. 7-10. The replacement sheets show this numeral in the place described for it in the text. Therefore, this correction introduces no new matter prohibited by 35 USC §132.

§103 Rejection of the Claims

Claims 1, 4, 5 and 12-29 were rejected under 35 USC § 103(a) as unpatentable over Dundas et al. (Proceedings of the 1997 International Conference on Supercomputing, pp. 68-75 (1997)) in view of Ukai et al. (U.S. 5,983,324). Applicant respectfully traverses these rejections.

Applicant agrees with the Office action that Dundas shows a run-ahead system but has no suggestion of protecting the pre-executed cache data against eviction. In fact, Dundas appears not to evict cache lines at all during run-ahead speculative execution. Dundas states only that cache misses during run-ahead are placed in a memory access queue; p. 68 col. 2. Cache misses during run-ahead are the only reason to evict lines that are already in the cache. Thus, Dundas' system would not benefit from protection against eviction. [1]

Ukai's situation differs considerably from Dundas'. Ukai's data-streaming system will crash completely if an overflow causes data not yet consumed by one user program to be

[1] --- Even if Dundas were to evict during run-ahead, the need for protection cannot be assumed, because lack of protection will not lead to failure of a run-ahead system. Rather, as the Specification states on page 2, lines 25-27, "[O]verflow detracts from the benefit provided by the run-ahead prefetcher." That is, Applicant provides an enhancement to a feature, rather than fixing a possible failure mode. Dundas' teaching does not suggest to one skilled in the art even the possibility that overflow might occur during run-ahead, much less that run-ahead performance might be further increased by reducing its occurrence. Thus, Dundas offers no motivation for an artisan to seek an enhancement anywhere for the opportunity that Applicant has recognized.

overwritten by data accessed for another user program. Ukai replaces multiple fixed-size sequential-access FIFO[2] data buffers with a single buffer pool shared by multiple programs. The advantage is that some programs can use more than an aliquot portion of the total pool, as long as others use less than their share, or when fewer programs are requesting sequential data. But this flexibility has a price. The user programs operate independently of each other, without any overall coordination of data accesses. Therefore, the entire pool can overflow, and data will be actually overwritten within the buffer pool. Conventional single-program FIFO buffers have long employed protection against placing additional data into a full buffer before the user has consumed some of it. Ukai's invention merely extends this concept by providing an ad-hoc control mechanism that operates over multiple buffers that share a single block of address space. That is, Ukai himself generates a need for protection, because of his unique buffer-pool configuration. Accordingly, one skilled in the art would not be led by any of Ukai's disclosure to apply a feature of Ukai's multi-program system for accessing sequential data streams from a disk drive to Dundas' system for speculative execution of instructions while a processor cache awaits a memory load after a cache miss.[3]

Accordingly, there is no motivation for combining Dundas and Ukai from either direction. Dundas neither suggests nor requires protection from cache overflow in order to operate correctly; Ukai needs protection only because of the unique configuration of his buffering system, where multiple independent programs share a common pool. The purposes of these two systems are so far apart---multi-program sequential data buffering (Ukai) versus pre-

---

[2] --- first-in, first-out, sometimes called "silo" buffers.

[3] --- The Office Action also conflates the two different methods by which instructions/data enter the cache/buffer, calling both of them "prefetch" (e.g., par. 3). When Applicant (or Dundas) "pre-executes" instructions, these instructions are already in the cache, and not, as in Ukai, in an external storage device awaiting transfer into the buffer area 10, Fig. 1. "Pre-fetching" in a run-ahead system (Applicant's or Dundas' or others') does not actually move instructions from main memory into the cache during run-ahead mode. This would defeat the purpose run-ahead. Run-ahead only occurs while the cache is already waiting for a cache miss to be resolved by accessing the much slower main memory. To access further instructions from the slow main memory during run-ahead would further slow the processor's operation. If an instruction that is pre-executed causes another cache miss, the prefetcher only designates certain main-memory addresses to be accessed after the processor returns to normal mode. Ukai, in contrast, always actually moves data from the device to the buffer, and neither has nor needs "modes" to differentiate between mere designation and actual transfer. (Also, of course, Ukai never pre-executes or otherwise re-uses any data already in his buffer area 10.)

execution processor cache[4] (Dundas) that motivation to combine these references, or to modify Ukai to shoehorn its protection into Dundas, must be found in the references themselves. While a reference may have an umbra based upon what it suggests to an artisan,[5] the much more recent holding in *In re Lee*, 61 USPQ2d 1430 (CAFC 2002) requires motivation in the corners of the references themselves to make the combination in the first place. In this case, the motivation for combining these two disparate references springs only from Applicant's disclosure, and not from the references themselves.

As to claim 1, the Office Action admits that Dundas does not teach "setting a protection bit" associated with a cache line. In fact, Dundas teaches no cache-line protection at all, in any form.

Ukai, on the other hand, has no suggestion of a normal mode in which "instructions" are "executed from a cache." Ukai only delivers sequential data to a user program from a buffer,[6] and does not execute any kind of instructions from his area 10. Claim 1 further recites a run-ahead mode; as a threshold issue, Ukai does not have two different "modes" in which data are accessed any differently from one mode to the other. Further, Ukai has no suggestion of "future instructions," or instructions that are executed but that "do not produce valid results." These added claim recitations do not narrow the scope of claim 1; they merely incorporate the definition of the modes already in the Specification, e.g., at page 6:11-15.[7] These recitations make explicit the wide differences between the systems of Ukai and Dundas, which, as urged above, prevent their combination under 35 USC 103.

---

[4] ---Although Ukai calls his buffer pool a "cache," it does not operate at all like processor cache. A cache permits a sustained processor speed exceeding the sustained access rate of the slower main memory because the processor can reuse data items in the cache multiple times. The purpose of a buffer , on the other hand, is merely to smooth out speed variations in the device that provides data and/or the device that consumes data. The provider provides each piece of data only once, and the consumer uses each piece of data only once, so that the sustained data rate cannot exceed the speed of the slowest device.

[5] --- *In re McLaughlin*, 170 USPQ 209 (CCPA 1971) and *In re Bozak*, 163 USPQ 545 (CCPA 1969), cited on page 7 of the Office Action.

[6] --- As pointed out above, calling Ukai's area 10 (Fig. 1) a "cache" is not accurate.

[7] --- This colon notation denotes page and line numbers in the Specification; e.g., "page 6:11-15" refers to page 6, lines 11-15.

Dependent claims 4 and 5 incorporate all the recitations of their parent claim 1. As to claim 4, the Office Action asserts that Ukai teaches "protecting only prefetched data" (page 3). What claim 4 actually says, however, is that when normal execution starts, the protection for the prefetched instructions goes away: "clearing all protection bits," which are associated only with prefetched instructions. Ukai never cancels protection on his data until they are actually used, whereas Applicant does not actually execute any cache instructions (i.e., "valid" instructions) until resuming normal mode, and, by the language of claim 4, after its protection has been stripped away. One rationale for this in some embodiments is that, if the cache misses again after resuming normal mode, a main-memory address that it requires immediately should take precedence over prefetched instructions that might or might not become valid at a later time. Claim 5 includes a similar recitation: all protection bits for prefetched data are cleared "upon starting run ahead execution." Thus claims 4 and 5 distinguish over even an improper combination of Dundas and Ukai.

Independent article claim 29 includes the method recitations of claim 1, so that combining the Ukai patent with the Dundas publication is equally improper, for the reasons given in connection with claim 1. Dependent claims 30-31 incorporate all the recitations of their parent claim 29.

Independent claim 12 is amended to include recitations explicitly defining the normal and prefetching (run-ahead) threads as in claim 1, above. Here again, Dundas teaches no "protection bits," or indeed protection of any kind. Ukai has no suggestion of any threads that "execute ... instructions from a cache;" he only consumes data from a buffer. Claim 12 further recites another mode, not suggested in Ukai, that "speculatively executes only future instructions that do not produce valid results." Ukai has no suggestion of speculative or run-ahead execution of any kind, much less of such execution from his area 10. As in claim 1, the Office Action applies the term "prefetching" to Ukai for actual movement of data into a buffer, while Applicant's run-ahead prefetching either executes instructions already in the cache or merely designates a memory address for possible future access, quite a different type of operation.[8] These recitations

[8] --- Dundas describes a type of "prefetch" for run-ahead on page 68, col. 2: "The cache misses generated during runahead are placed in a data memory access queue, DMAQ." As noted in connection with claim 1, actually performing these accesses during run-ahead mode would defeat the advantage of this mode.

make explicit the wide differences between the systems of Ukai and Dundas, which, as urged above, prevent any combination permissible under 35 USC 103.

Dependent claims 13-15 define over the references for the same reasons. Claim 13 distinguishes even an improper combination, for the same reasons as do claims 4 and 5 above. As to claim 14, Applicant does not understand how the comments in the second full paragraph of page 3 in the Office Action pertain to anything found in the references. Dundas says nothing about any kind of threads at all, and Applicant finds nothing in Ukai suggesting tying prefetching to a "predetermined section of code" in any of Ukai's user programs. In fact, this would seem to undermine Ukai's purpose. Likewise, Applicant finds nothing that remotely suggests how the "optimizing compiler" of claim 15 might serve any purpose in Ukai's system.

Apparatus claim 16 is amended to explicitly recite the execution logic[9] that drives the other elements. Claim 16 then explicitly recites the normal and run-ahead modes as in claims 1 and 12. Here again, Ukai only accesses data, and does not suggest executing "program instructions ... from the cache" area 10 in his Fig. 1. Further, the recitation of a run-ahead mode in which "only future instructions that do not have valid results" point out the wide gulf between Dundas' run-ahead cache and Ukai's multi-program data buffer that renders the combination of these two references improper under 35 USC 103. Claims 17-20 distinguish the references for the these reasons and others.

Apparatus claim 21 is amended to sharpen the existing recitation of "speculative execution" as happening during a run-ahead mode which executes "only invalid instructions and produces invalid results." Ukai cannot be said to teach a prefetch method that loads "'data to be executed'" (Office Action, top of page 4). Calling Ukai's consumption of data by the user programs "execution" is purely an importation from Applicant's disclosure. Ukai's transfer of data to a user program is wholly unlike executing an instruction. Further, of course, Ukai hasd no suggestion of "speculative" execution of any kind---that is, execution involving "only invalid instructions" and "invalid results." Moreover, Ukai does not have a "plurality" of caches; the whole point of his system is to combine the buffering of data for multiple programs in a *single* area 10, where one program may use more of the area when fewer programs are running, or when the other programs do not require as much data. Here also, then the claim includes

[9] --- See page 6:4 of the Specification.

recitations that point out how very different the system of Ukai is from that of Dundas---so different that an artisan desiring to design a run-ahead cache would not even look at Ukai without prompting from Applicant's invention. This is improper under 35 USC 103, as explained in *In re Lee*, cited above. Claims 22-24 depend from independent claim 21.

Claim 25 is amended similarly to claim 1 above to define the already-recited normal and run-ahead modes explicitly. The run-ahead mode "speculatively executes" instructions, whereas Ukai has no speculation of any kind, but rather pulls into his area 10 only data from a file that has already been definitely requested by a user program, and discharges data from the area only when that data has been definitely consumed by the user program. Ukai further has no mention of or motivation for instructions "that do not produce valid results." For the reasons stated above, then, the systems of Dundas and Ukai are so far apart in their purpose and operation that they can be combined only in the light of Applicant's disclosure. Dependent claims 26-28 depend from parent claim 25.

Claims 2, 3, 6-11, 30 and 31 were rejected under 35 USC § 103(a) as unpatentable over Dundas et al. in view of Ukai et al. and further in view of Patrick et al. (U.S. 5,920,889). Applicant respectfully traverses these rejections.

Claims 2-3 depend from claim 1. The Petrick patent shows does not show the run-ahead mode of Dundas, and has as little connection with Ukai as does Dundas. Therefore, Petrick adds nothing, and the combination of Dundas with Ukai has been demonstrated above to be improper under 35 USC 103.

Claims 30-31 depend from claim 29, which was rejected under an improper combination of Dundas and Ukai, to which Petrick adds nothing.

Independent claim 6 recites "evicting" a potential victim. As noted above, Dundas appears not to employ eviction at all, in which case protection has no purpose. While Dundas recognizes that run-ahead may produce further cache misses, he seems content to place their addresses in a data memory access queue (page 68, col. 2) for action after run-ahead completes, when the original cache miss has been resolved. Grafting Petrick onto Dundas thus actually *creates* the problem that Applicant ameliorates. Combining one reference with another to create a problem, and then adding a third reference to solve the problem thus created is an unlikely path

for one skilled in the art, and is surely improper under 35 USC 103. Claims 7-8 depend from claim 6, and incorporate all its recitations.

Claim 9 has a situation parallel to that of claim 6. Here again, there is no reason to replace a cache line until a cache miss occurs in run-ahead mode. Dundas appears not to actually "replace" such lines, but, from Dundas' teaching, one skilled in the art learns only that he queues them up for further action after run-ahead has completed. No replacement, no benefit in protection. Combining Ukai at this point is bootless, because protection would provide no advantage at all to Dundas. And, as noted above, adding Petrick and then folding in Ukai requires one to first create a problem and then to seek a solution.[10] Dependent claims 10-11 include the recitations of parent claim 9.

[10] --- Also, as noted elsewhere, employing eviction would not cause Dundas' system to fail, but only make it run more slowly than it would with protection.

Conclusion

For the above and other reasons, Applicant urges that the claims meet all statutory requirements, and respectfully requests reexamination and allowance thereof. The Examiner is invited to telephone Applicant's attorney at (612) 373-6971 to facilitate prosecution of the Application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 19-0743.

Respectfully submitted,

CHRISTOPHER B. WILKERSON

By his Representatives,

SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
Attorneys for Intel Corporation
P.O. Box 2938
Minneapolis, Minnesota 55402
(612) 373-6971

Date 16 July 2004

By _____
J. Michael Anglin
Reg. No. 24,916

CERTIFICATE UNDER 37 CFR 1.8: The undersigned hereby certifies that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail, in an envelope addressed to: Mail Stop RCE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this 16 day of July, 2004.

KACIA LEE
Name

Kacia Lee
Signature